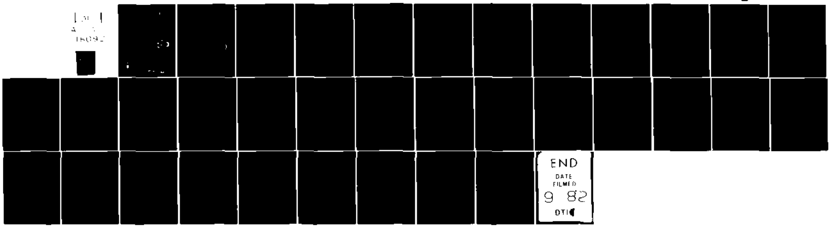


ARO 16381.6-EL

(12)

DESIGN METHODOLOGY FOR MULTIPLE MICROCOMPUTER ARCHITECTURES

Department of Electrical and Computer Engineering

New Mexico State University

NMSU-TR-82-1

Final Report For

Grant [illegible]

Submitted to:

U. S. Army Research Office
Box 12211
Research Triangle Park, NC 27709

July 1982

DESIGN METHODOLOGY FOR MULTIPLE MICROCOMPUTER ARCHITECTURES

Department of Electrical and Computer Engineering

New Mexico State University

NMSU-TR-82-1

Final Report For

Grant DAAG29-79-C-0100

Submitted to:

U. S. Army Research Office
Box 12211
Research Triangle Park, NC 27709

July 1982

DISCLAIMER

The findings of this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NMSU-TR-82-1	2. GOVT ACCESSION NO. AD-A118092	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Design Methodology for Multiple Microprocessor Architectures		5. TYPE OF REPORT & PERIOD COVERED Final Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Dr. G. M. Flachs, G. M. Chavez, B. N. Malm		8. CONTRACT OR GRANT NUMBER(s) DAAG29-79-C-0100
9. PERFORMING ORGANIZATION NAME AND ADDRESS New Mexico State University Las Cruces, NM 88003		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Research Office Box 12211 Research Triangle Park, NC 27709		12. REPORT DATE July 1982
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Design Methodology
Multiprocessor Design
Concurrent Processing Model
Process Model
Multimicro Architecture
Fault Detection Model
Real-Time System Design

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A mathematical model is developed for designing and analyzing multiple microcomputer architectures. The model combines a user application algorithm, a generalized distributive operating system, and a generalized multiprocessor architecture into an integral system representation. When applied to real-time design problems, it provides a method to establish the processing and communication resources required to meet real-time constraints.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

There is a rapidly growing interest in applying multimicro architectures to solve real-time processing problems. Tremendous computing power is available from multiple microcomputer architectures; however, it is difficult to realize this power with current design methodologies. There are several reasons why designers find it difficult to realize multimicro architectures. First, the design of these architectures requires new problem formulation concepts and analysis tools to utilize concurrent processing techniques. Secondly, there is a need for a better practical and theoretical understanding of the performance that can be achieved with multimicro architectures in the real-time environment. Thirdly, there is a need for better hardware and software design tools for multimicro architectures. Finally, more research and experience is required to insure performance, reliability, and maintenance of these architectures.

Our research focused the art of designing and applying multimicro architectures on three major topics. These topics are:

- a multimicro design model
- a generalized multimicro architecture
- a multimicro fault diagnostic model.

The first problem concerns the development of mathematical foundations for modeling, designing, analyzing, and applying multimicro architectures to real-time processing problems. A new mathematical model [3, 19] is introduced to model and analyze the dynamic behavior of a user application on a multimicro architecture. The model combines a user application algorithm, a generalized distributed operating system, and a generalized multiprocessor architecture into an integral representation of the system. The user application is represented by a concurrent network of user processes. Each user process consists of the enabling conditions, the resulting conditions, and a sequential program with associated data structures. These processes are defined so that the rich parallelism is between the processes rather than within them. The essential functions of a distributed operating system are included in the model to direct the execution of the user processes among the available processing and communication resources. These operating system functions include process scheduling and resource management activities. Finally, the multiprocessing architecture is represented as a constrained set of processing and communication resources in the model. When the model is applied to a user real-time processing problem, the analyst can establish the appropriate processing and communication resources required to meet real-time processing constraints.

The second problem concerns the development of a generalized multimicro architecture and the establishment of theoretical and practical performance properties. The basic structure of the architecture is derived from our mathematical model for multimicro architectures which explicitly relates the user concurrent algorithm and a concurrent operating system to the multimicro architectures. The architecture [18, 20] consists of a loosely connected set of multimicro clusters. Each cluster is capable of stand-alone operation or working as an integral part of a collection of clusters. The operating system is distributed among the processing units in the architecture. The processing units are interconnected by a local communication system within a cluster and a global communication system between clusters. Each communication system consists of a collection of independent, directly connected, and shared buses. The bus protocol supports a multiaccess broadcast protocol similar to Ethernet communications. The physical buses can be realized with a simple two-wire RS-232 bus using standard UARTS and bus drivers, or for a higher performance the buses can be realized with an Ethernet communications network. The architecture is modular and uses standard processing modules that are commercially available. The decentralization of the operating system greatly simplifies the software architecture to a set of standardized software modules that are realized with programmable read-only memory modules. A dynamic concurrent multiprocess model is presented for the generalized multimicro architecture that provides design and performance guidelines for real-time applications.

The third problem concerns the development of an analytical model for characterizing the dynamic behavior of faults in a multimicro architecture. An analytical Fault Diagnostic Net (FDN) model [10, 15] is formulated to perform on-line fault diagnostics for real-time processing problems. Using a built-in-test (BIT) processor to monitor and analyze a processor's internal bus activity, the model characterizes the relationships between the recorded activity and the faults to be detected. Algorithms and theorems are given for designing diagnosable multimicro architectures.

A summary of the major results of the research is presented in Chapter 2. A list of the personnel supported with degrees granted is given in Chapter 3. A list of publications, technical reports, and dissertations resulting from the grant is given in Chapter 4.

II. SUMMARY OF MAJOR RESULTS

Recent VLSI advances in microprocessor technology is the driving force motivating the application of multimicro architectures to real-time processing problems. The primary objective of multimicro architectures is to utilize the attractive computing power, size, and cost/performance ratio of the VLSI chips available. The approach currently used to design multimicro systems is to select an attractive architecture, design the hardware architecture and interconnection network, design the software architecture, and integrate the hardware and software architectures to realize the system. When the system is built, performance tests are made to verify the performance and reliability of the system. Expensive and time-consuming design changes in hardware and software are often necessary. The resulting systems often have unique hardware architectures that are difficult to document and maintain. Each design is a one-time experience, and little knowledge and design capability is carried over to other problems. This inability to transfer multimicro design knowledge is true both in industry and in university environments. In the industrial environment, it reduces productivity and increases the cost of products. In the academic environment, it greatly reduces the research creativity and productivity. The key to sharing multimicro design knowledge is in the development of mathematical foundations for designing and analyzing multimicro architectures.

System modeling techniques, operating system concepts, concurrent algorithm models, and computer architecture concepts are combined with parallel processing methods to establish an analytical basis for modeling, analyzing, and designing multimicro systems. A concurrent multiprocessor model is introduced for designing and analyzing the performance of multimicro architectures. The model integrates the functional behavior of a distributive operating system and a generalized multimicro architecture with the user real-time concurrent algorithm to characterize the dynamic behavior of the system.

2.1 A Multimicro Design Model

A real-time concurrent algorithm is viewed as a set of processes along with a set of constraints which define their order of execution. The processes are defined so that the rich parallelism is between the processes rather than within them. A process consists of enabling conditions, resulting conditions, conditional test, private data structure, and a structured IF-THEN-ELSE sequential program.

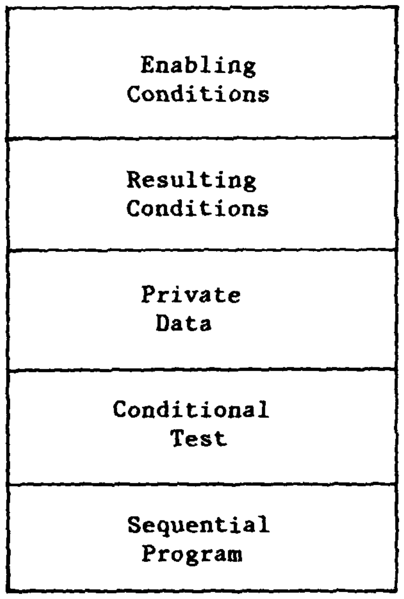


Figure 1. Process Structure

The enabling conditions are the constraints that must be satisfied before the process can execute. The resulting conditions are the results that are sent to enable other processes. The private data define the data structures used by the process. The sequential program has an IF-THEN-ELSE structure which is controlled by a conditional test. The structure of the sequential program is shown in Figure 2.

```
ACTIVATION (OS)
----------------------
IF CONDITION
    THEN
      TRUE PROCEDURE
    ELSE
      FALSE PROCEDURE
ENDIF
----------------------
TERMINATION (OS)
```

Figure 2. Sequential Program Structure

The activation functions are performed by the operating system which activates and links the data structures to the process. Depending upon the condition, either the true or false procedure is executed. During the process, execution results are formulated as output messages and placed in the output buffer. The termination functions are again performed by the operating system, which terminates the process and sends the messages stored in the output buffer. The key idea is to place the burden of data linking and message sending on the operating system rather than on the process. This greatly reduces the effort required to write the process code, and it makes the process code independent of the operating system and the hardware environments.

Task Model

An extended dynamic Petri model is introduced to model the processes and their interrelations. This model is called the Task Model. The Task Net consists of a structured interconnection of transitions and places. Transitions represent the processes and define the enabling conditions, represent process execution, and define the result conditions. The places represent information points and define the flow of information and process constraints in the network.

By defining a finite and bounded execution time for each process, it is possible to establish the dynamic behavior of the network in terms of the motion of tokens in the model. With this interpretation, the presence of a token in a place implies that the information is available, and the presence of a token in a transition implies that the process is currently executing. Since many transitions can simultaneously have tokens, the model has the ability to describe concurrent processing behavior.

The dynamic behavior of Task Net is generated by the execution of processes at the transitions. A transition can be initiated if and only if the transition is idle (no token present) and all input arcs to the transition come from places containing at least one token (information ready). When a transition is activated, one token is taken from each input place (places which have an inout arc to the transition), and one token is placed in the transition to indicate the process is executing. A control place, represented by a demand symbol, is associated with a transition to direct the process execution to the true process procedure if a token is present or to the false process procedure if a token is not present. The presence or absence of a token at the control place does not affect the enabling of the process. When the transition is activated, a token is taken from the control place if one is present. At the completion of the process execution, the token is removed from the transition, and one token is placed in each output place specified by the transition.

The structure of a transition is shown in Figure 3. The input places $\{P_1,P_2,P_3\}$ are the transition input places, and they define the enabling conditions for the transition. The control place C_1 directs the process execution to either the true or false procedure. The output places P_4 and P_5 are results from the true procedure. The output places P_6 and P_7 are results from the false procedure.

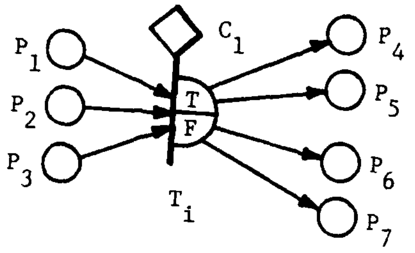
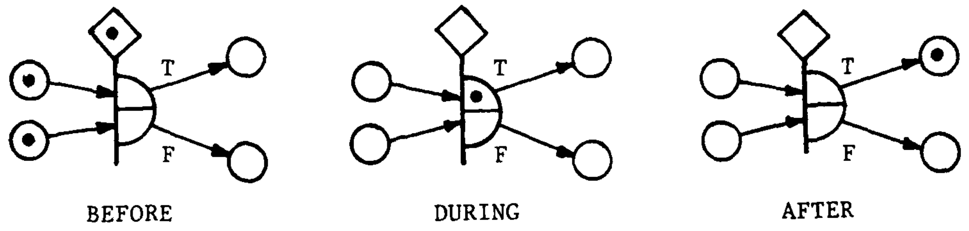
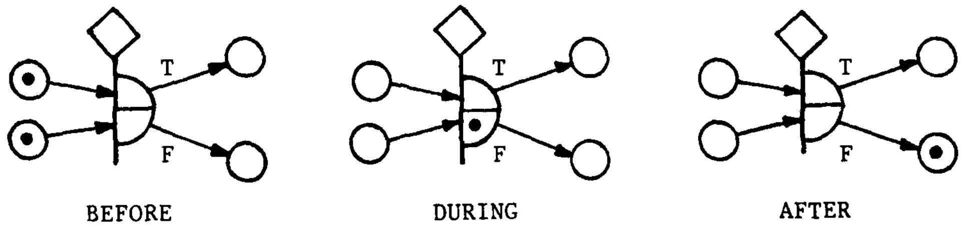


Figure 3. Transition Structure

The process of executing a task is shown in Figure 4.



4(a)



4(b)

Figure 4. Transition Execution

A nonlinear discrete system model is developed to establish and analyze the dynamic behavior of the Task Model. The model is developed in matrix form using standard APL matrix operators. The dyadic APL operators $o \in \{=,>,<,\neq\}$ on $n \times m$ matrices A and B are used to define a binary $n \times m$ matrix $A \; o \; B$ whose entries are 1 iff $a_{ij} \; o \; b_{ij}$ for each i and j.

The Task Model is a directed graph $G = (T,P,A)$ of transitions (processes), places (information points), and directed arcs with a discrete system of execution rules for moving the tokens in the graph at integral time units. The graph has a finite set of transitions $T = \{T_1,T_2,...,T_n\}$, a finite set of information places $P = \{P_1,P_2,...,P_m\}$, a finite set of control places $CP = \{CP_1,CP_2,...,CP_n\}$, and a finite set of arcs $A = (TxP)\cup(PxT)$. The nodes of the graph are defined by $N = T\cup P$. A marking of the model is defined by three mappings:

$$MT : T\rightarrow\{0,1\}$$

$$MP : P\rightarrow I = \{0,1,2,...\}$$

$$MC : CP\rightarrow I = \{0,1,2,...\}$$

which assign tokens to the transitions (MT), information places (MP), and control places (MC) respectively. The dynamic behavior of the markings with respect to time plays an important role in the design of concurrent processing systems.

The tokens in the Task Model move only when transitions are enabled and completed. For a given transition T_i, the set of input places (IP), the set of true output places (TOP), the set of false output places (FOP), and the control place (CP) are defined by:

$$IP(T_i) = \{P : (P,T_i)\varepsilon A\}$$

$$TOP(T_i) = \{P : (T_i,P)\varepsilon TA\}$$

$$FOP(T_i) = \{P : (T_i,P)\varepsilon FA\}$$

$$CP(T_i) = CP_i$$

where $TA \subseteq TxP$ is the set of true output arcs, $FA \subseteq TxP$ is the set of false output arcs, and $TA \cup FA=TxP$.

A transition T_i is initiated with marking MT and MP iff $MT(T_i)=\emptyset$ and $MP(T_i)>\emptyset$ for all $p\varepsilon IP(T_i)$. When a transition T_i is initiated, new markings are generated by:

$$MT(T_i) = 1,$$
$$MP(P_j) = MP(P_j)-1 \text{ for all } P_j \varepsilon IP(T_i),$$
$$\text{and} \quad MC(T_i) = \text{Max } \{\emptyset, MC(T_i)-1\}.$$

When a transition T_i completes execution, the new markings are defined by:

$$MT(T_i) = \emptyset$$

$$\text{and} \quad MP(P_j) = \begin{cases} MP(P_j) + 1 \text{ if } P_j \varepsilon TOP(T_i) \text{ and } MC(T_i)>\emptyset \\ MP(P_j) + 1 \text{ if } P_j \varepsilon FOP(T_i) \text{ and } MC(T_i) = \emptyset \\ MP(P_j) \quad \text{else.} \end{cases}$$

The Task Model is a system of execution rules for moving tokens in the graph and a set of vectors $R = (\vec{K}, \vec{C}, \vec{V}, \vec{D}, \vec{S}_0, \vec{S})$. The vector $\vec{K}(\tau) = (K_1(\tau), K_2(\tau),\ldots,K_m(\tau))$ define the number of tokens at the information places at time τ. The vector $\vec{C}(\tau)=(C_1(\tau),C_2(\tau),C_2(\tau),\ldots,C_n(\tau))$ defines the number of tokens at the control places at time τ. The vector $\vec{V}(\tau)=(VT(\tau),VF(\tau))$ defines the time left before completion of the true and false transition blocks. The vector $\vec{D}=(DT,DF)$ defines the processing times for true and false transition procedures. The state vector for the system is defined by $\vec{S}(\tau) = (\vec{K}(\tau), \vec{C}(\tau), \vec{V}(\tau))$ and $\vec{S}(0)$ is the initial state vector. The next state mapping $\delta : \vec{S}(\tau)\rightarrow\vec{S}(\tau+1)$ defines the system state $\vec{S}(\tau+1)$ at time $(\tau+1)$ in terms of the state $\vec{S}(\tau)$ at time τ constrained by the execution rules.

The next state mapping is defined as an ordered set of operations. The set of all enabled transitions is given by:

$$EN = \{T_i : T_i \varepsilon T, V_i=0, K_j>0 \forall P_j \varepsilon IP(T_i)\}$$

and the set $FR = PR(EN) \subseteq EN$ defines the transitions that are firable after conflicts are resolved by a priority function. The set of executing transitions is $EX = \{T_i : T_i \varepsilon T, V_i \geq 1\}$. The set of true and false transitions completing execution are given by $ZT = \{T_i : T_i \varepsilon T, VT_i = 1\}$ and $ZF = \{T_i : \varepsilon T, VF_i = 1\}$. In terms of these definitions the Task Model is defined by the following ordered set of operations:

1. Remove tokens from finished transitions and increment the proper output places. For each $T_i \varepsilon Z = ZT \cup ZF$

$$K_j(\tau+1) = K_j(\tau) + 1 \quad \text{if} \quad P_j \varepsilon TOP(T_i) \quad \text{and} \quad T_i \varepsilon ZT$$

$$\text{or if} \quad P_j \varepsilon FOP(T_i) \quad \text{and} \quad T_i \varepsilon ZF.$$

2. Decrement the processing times of all executing transitions

$$v_i(\tau+1) = V_i(\tau)-1 \quad \text{if} \quad T_i \varepsilon EX$$

3. Resolve conflicts between enabled transitions with a transition priority function

$$FR = PR(EN)$$

4. Initiate enabled transitions and decrement token counts in the input and control places. For each $T_i \varepsilon FR$

$$V_i(\tau+1) = \begin{cases} DT_i & \text{if} \quad C_i \geq 1 \\ DF_i & \text{if} \quad C_i = \emptyset. \end{cases}$$

and for each $P_j \varepsilon IP(T_i)$

$$K_j(\tau+1) = K_j(\tau)-1.$$

and $$C_i(\tau+1) = \text{Max}\ \{\emptyset, C_i(\tau) - 1\}$$

These equations form a nonlinear system of difference equations which are solved to establish the dynamic behavior of the state vector $\vec{S}(\tau)$ in terms of the initial marking $\vec{S}(0)$. The Task Model has important mathematical properties [19] that allow the behavior of the model to be studied without solving the nonlinear difference equations. One such property is the reachability tree that can be obtained from the structure of the Task Model. Deadly embraces and unbounded synchronization problems can be recognized and solved. The solution of the model establishes the total concurrency in the application algorithm since no operating system or architectural constraints are imposed by the model. If the response of the Task Model does not meet the real-time constraints, the designer must go back and reformulate the problem to expose more parallelism in the processes.

Process Model

A process model is introduced to model the activation, execution, and termination of a process on a generalized multimicro architecture under control of a distributed operating system. The process is constrained to be noninterruptable and nonrecursive. The hardware architecture and the operating system provide the environment that governs the dynamic behavior of the process. The process is modeled at a level of abstraction that specifies the tasks in an operating system associated with allocation, activation, execution, and termination of the process in a multimicro architecture. The process is modeled with the Task Model introduced in the previous section. This net of tasks is developed for a process and the process net is substituted for the application process in the concurrent algorithm. The resulting model is called a concurrent multiprocess system model. The substitution property allows the Task Model to be used at the application level as well as at the process level to provide proven designs and implementations on concurrent algorithms of multimicro architectures.

A process is modeled in a generalized way that focuses on the interrelationships between the process, the multimicro architecture resources, and the controlling operating system. Transitions in process model represent the tasks required to initiate, execute, and terminate a process. The architecture resources are modeled as semaphore places that define at any time the resources that are currently available. The architecture [20] consists of a loosly connected set of multimicro clusters. Each cluster consists of a tightly connected set of multimicros over a local communication bus structure. The clusters are connected by a global communication structure. Both local and global communication systems consist of a collection of independent, directly connected, and shared buses. The bus protocol supports a multiaccess broadcast protocol similar to Ethernet communications. The semaphores that define the architectural resources are:

- WPS is the number of working processors in a cluster
- KPS is the number of kernel scheduling processors in a cluster
- LCS is the number of local communication buses in a cluster
- GCS is the number of global communication buses.
- CLS is the number of clusters in the architecture.

These architectural semaphores are defined by the initial marking of the Task Model. Consequently, they are easily changed by the analyst to establish the architectural resources required for the given user application.

The detailed structure of the process model is given in references [19, 20]. A block diagram is given in Figure 5 to define the major functions present in the process model. When the process enabling conditions are satisfied, the process allocation subnet performs the operating system scheduling functions required to find an available working processor using the working processor semaphore. When the process allocation is complete, the operating system scheduler obtains the required communication semaphores and sends an activation message to the kernal operating system in the working processor. When the working processor receives the activation message, the process is loaded and execution begins. When the process is finished, the results are in a message format and stored in an output buffer. The communication subnet defines the functions required to obtain the communication semaphores and send the messages over the local and global communication buses. The process is then deactivated and tokens (messages) are sent to other processes by the scheduling operating system.

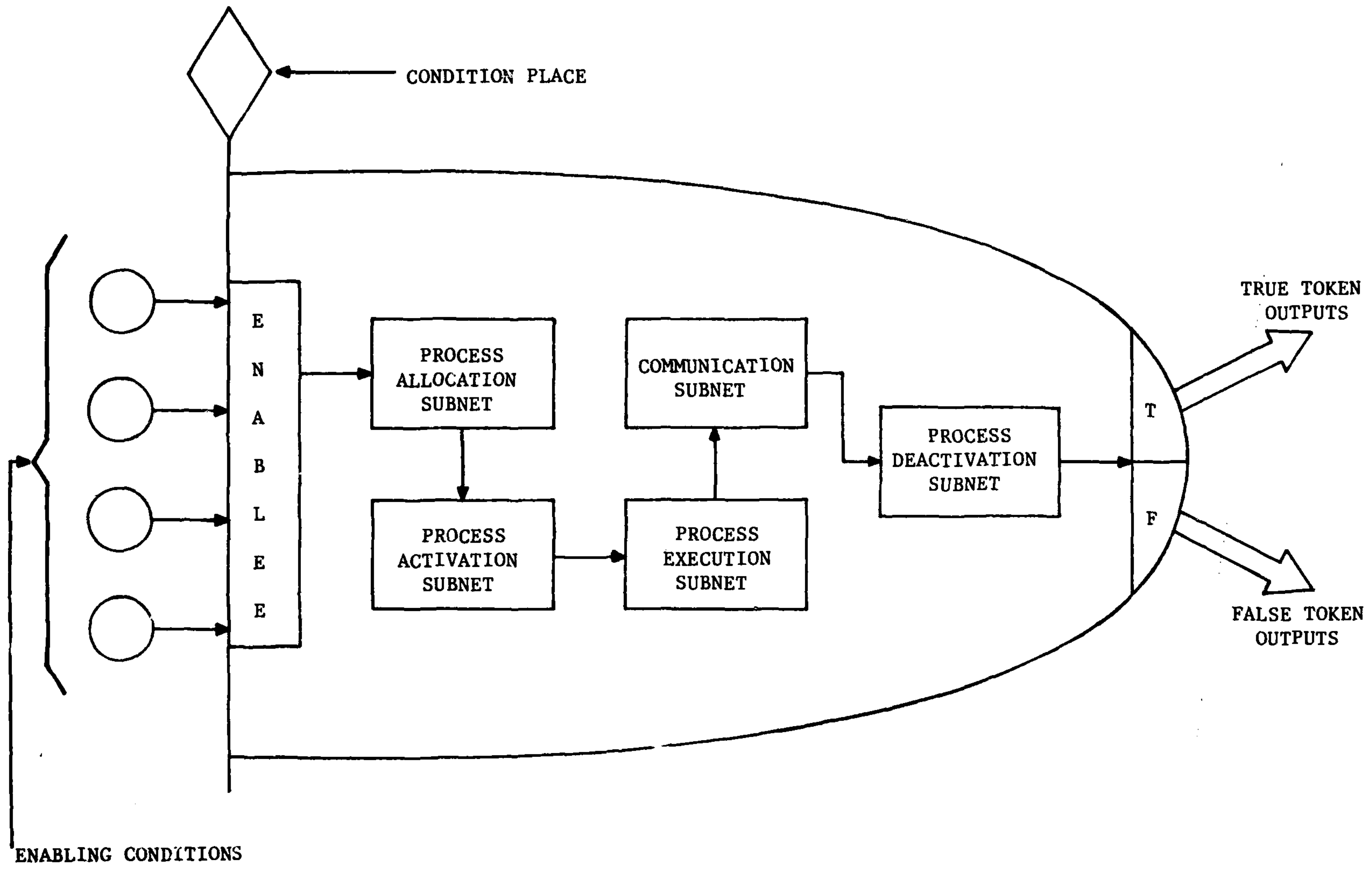


Figure 5 - Process Model

At this point, it is important to observe that operating system, processing, and communication resources are modeled as resource places (semaphores) in the process model. The quantity of the resources are easily specified by the initial marking of the Task Model, and they do not require structural design changes in the model. The quality (speed) of the resources are specified by the delay times in the transitions. Consequently, the analyst has a great deal of architectural flexibility at a parameter level for designing multimicro architectures. This leads directly into parameter optimization techniques for optimal design [20].

Concurrent Multiprocess System Model

At the design application level, a concurrent real-time algorithm is defined in terms of a set of potentially parallel processes and a set of constraints which define the communication structure between the processes. The Task Net allows the designer to formulate a real-time concurrent algorithm with standard structured programming techniques independent of operating system and architecture concepts. The topology of the network defines the process communication structure, the transitions define the processes, and the places define information and synchronization points.

To establish the Task Model, the designer estimates the number of instruction cycles required to perform the processes and selects an instruction cycle time for the level of technology desired for the application. At this point, the Task Model can be analyzed using a reachability tree to locate deadlocks or unbounded synchronization problems. If the net is bounded (necessary for practical realization), the potential markings can be established and potential concurrency analyzed. By solving the model the dynamic behavior can be analyzed to verify that the real-time processing constraints are potentially achievable.

The concurrent multiprocess system model is obtained by substituting the process model for each user process. This is strictly a mathematical substitution operation if the process model presented is acceptable to the designer. The resulting model includes the dynamic functional effect of the operating system and architectural constraints. By initializing the operating system and architectural semaphores, the model can be analyzed with the reachability tree to establish potential problems. By solving the model, the dynamic behavior of the system is established for the given operating system and architectural resources by providing virtually infinite architectural resources,

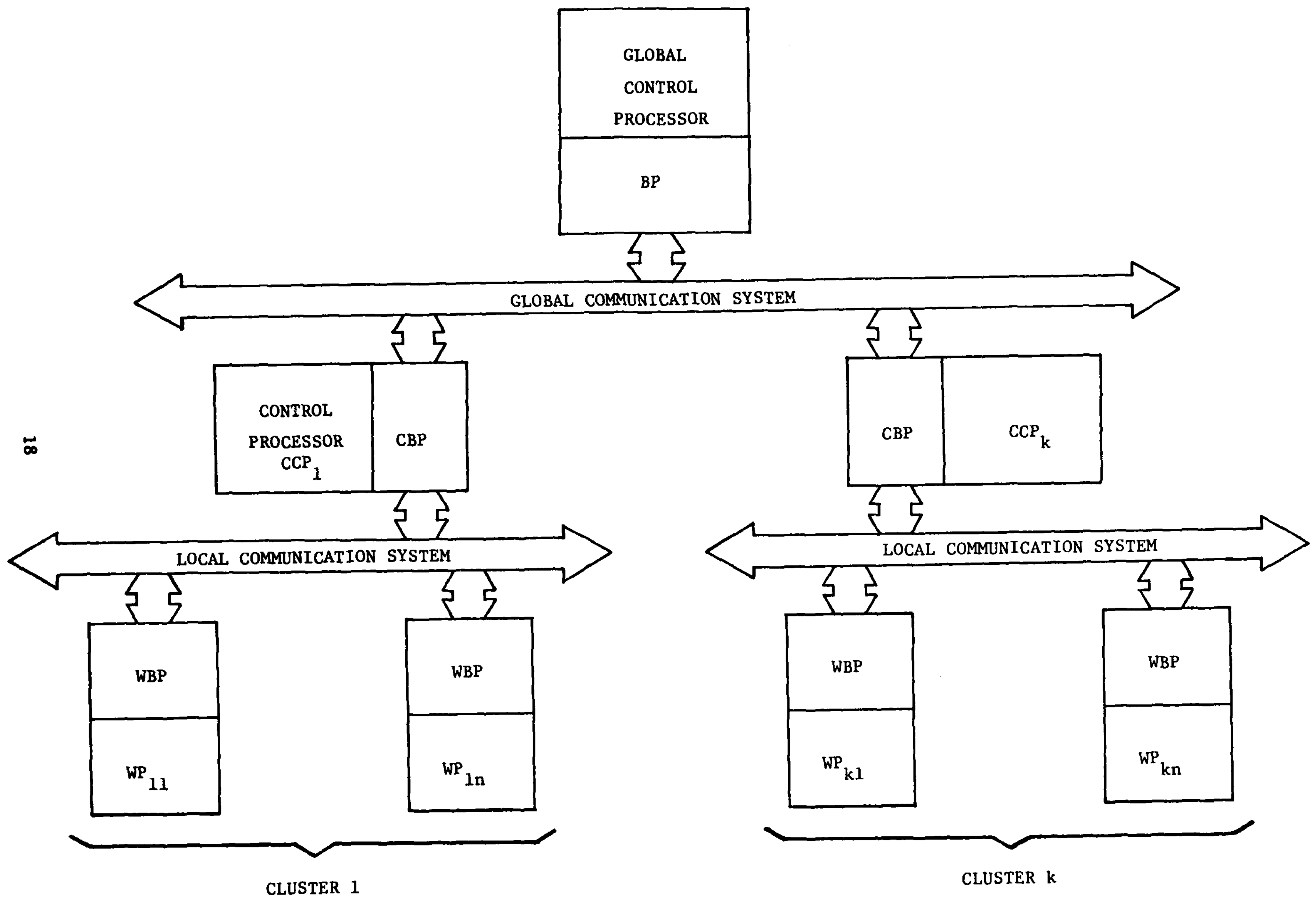


Figure 6 - Generalized Multimicro Architecture

the analyst can establish the timing constraints imposed by the operating system. Consequently, the analyst can decompose the timing constraints into application algorithm, operating system, and architectural constraints. This provides a powerful tool to locate and solve processing bottlenecks.

2.2 A Generalized Multimicro Architecture

A generalized multimicro architecture [20] is introduced to support the process model. The architecture, as shown in Figure 6, consists of clusters of working processors under the supervisory control of a global control processor. The global control processor is responsible for allocating and loading the processes into the clusters. The cluster control processor is responsible for scheduling and synchronizing the processes within a cluster. The working processors perform the process execution. The basic idea is to partition a large concurrent algorithm into loosely coupled process blocks and assign the process blocks to the clusters in the architecture. All interprocess communication information is contained in the process description. When the processes are allocated and loaded into the clusters, the global control processor signals the cluster control processors to start execution.

The global communication system connects the global control processor with the cluster processor. The local communication system connects the cluster control processor to the working processors in the cluster. Each communication system consists of a collection of independent, directly connected, and shared buses. The number of global and local communication buses are design parameters. The bus communication bandwidths are parameters that define the processing times associated with the local and global communication transitions in the process model. The bus protocol is not currently modeled in the process model; however, it is a topic of a dissertation [21] that is currently in progress. Early results from this research and laboratory experiments suggest a multiaccess broadcast protocol similar to Ethernet communications. The physical buses can be realized [18] with a simple two-wire RS-232 bus using standard UARTS and bus drivers. For higher performance, Ethernet channels or synchronous parallel buses can be used to implement the global or local communication channels.

Bus Processor

Each processor connected to the architecture has a bus processor to support the multiaccess-broadcast protocol. The bus processor is responsible for realizing the protocol, sending messages, and receiving messages. To send a message, the master processor simply places the message in the output buffer memory (OBM) and sets the output flag (OF). The bus processor monitors the bus activity, acquires the bus, sends the message, and checks for any errors.

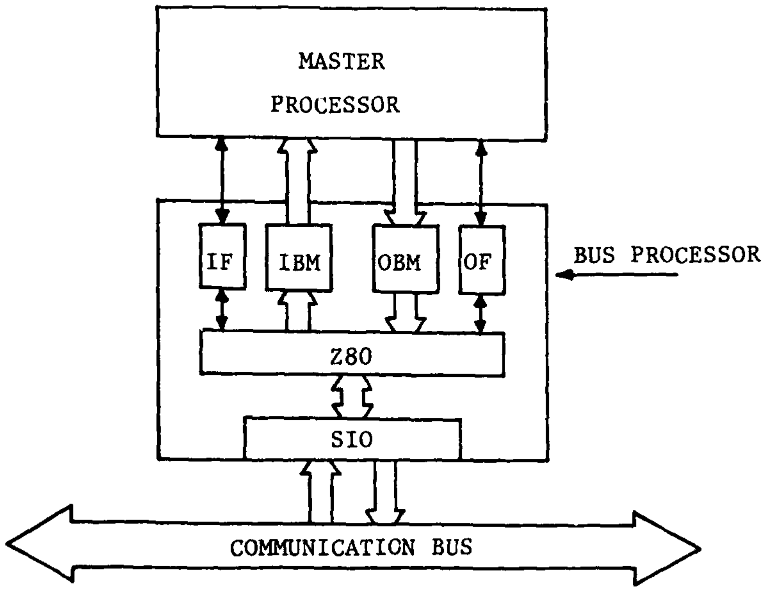


Figure 7. Bus Processor Structure

When a message is directed to the master processor, the bus processor receives the message, places it in the input buffer memory (IBM), and sets the input flag (IF). The bus processor is relatively inexpensive [18]; and it greatly reduces the master processor communication load.

The architecture has a modular structure that can be realized with a few standard modules. These modules include processing units, bus processors, and communication buses. The processing units can be realized with any single board microcomputer or minicomputer currently available. It is beneficial for the processing units to have substantial random access memories to reduce off-line storage problems. Consequently, the new sixteen-bit microprocessors are ideally suited for the architecture. The bus processors perform all communication in the architecture, and they are tightly connected to the processing units. Finally, the communication buses provide the desired information transmission medium.

The operating system for the architecture is distributed among the processing units. The global control processor is responsible for allocating, loading, and initiating the processes in the clusters. The cluster control processor is responsible for scheduling and synchronizing process execution in the working processors. The operating system in each working processor is responsible for executing the processes and handling communication with the cluster control processor. The bus processor communicates with its master processor and performs all communication functions.

Prototype Multimicro Architecture

A prototype version of the architecture [18] is being developed in our laboratory. The main purpose of the architecture is to provide a practical testbed to develop and test modeling techniques and operating system concepts.

The architecture is shown in Figure 8. The system uses a Z80 based microcomputer system as a control processor to perform both the global and cluster control functions. Z8000 single board microcomputers are used to realize the working processor clusters. The physical communication buses are realized with RS-232 twisted pair cables operating with the standard synchronous SDLC bit protocol. The baud rate of the channel can operate up to 800K baud rate. The network layer of protocol is common to all computers in the network. It is a hardware independent packet protocol similar to the X.25 standard. All messages between computers are segmented into blocks of bytes, called packets. There are control packets for sending short control messages and data packets for sending blocks of data. Each packet has a header which consists of (1) an ASCII start of text character, (2) a byte that defines the originator of the packet, (3) a byte that defines the receiver of the packet, (4) a control byte for sending control information and sequence numbers, and (5) a check sum error detection byte. At the network layer, acknowledgement messages are used to acknowledge the error-free reception of messages. If an acknowledge is not received in a given time interval, the packet is resent and the timer set again. This process continues until an acknowledgement is received or the packet is resent five times. When the packet is resent five times, without an acknowledgement, the communication channel is considered down.

A research effort [17] is also being conducted to apply the concepts developed in this report at the Holloman Air Force Base to loosely coupled minicomputers at the base. A high-level job control language is being developed to control clusters of HP1000 minicomputers for real-time data acquisition and analysis problems.

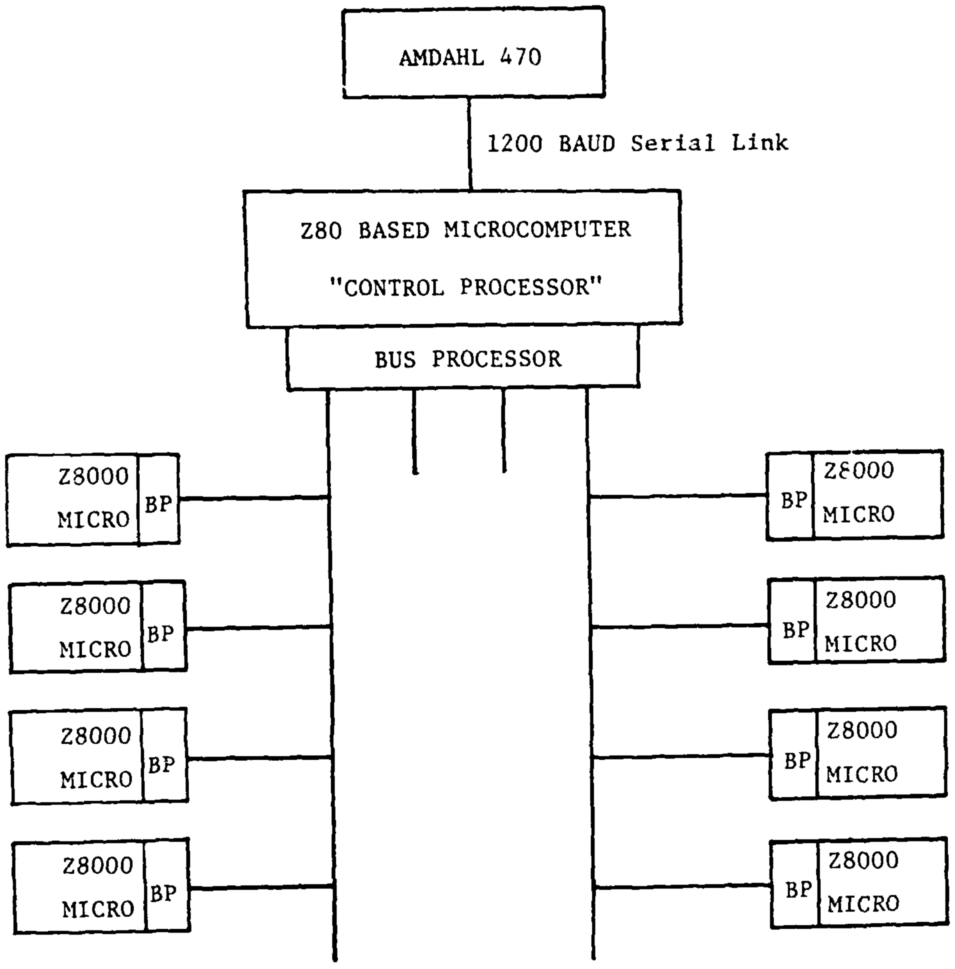


Figure 8. Prototype Multimicro Architecture

2.3 A Multimicro Fault Diagnostic Model

A real-time fault diagnostic model [10] is introduced to characterize the behavior of faults in a multimicro architecture. The basic assumption of the model is that the behavior of detectable faults can be observed on either the internal or external buses of the architecture. Built-in-test (BIT) processors are used to monitor the bus behavior of the multimicro architecture. Having access to the internal bus behavior of the working processors, the BIT processors records and analyzes the flow of instructions and data on the buses for possible errors. An analytical fault diagnostic net (FND) model is formulated to characterize the relationships between the recorded activity and the type of faults to be diagnosed.

Fault Diagnostic Model

The FDN model utilizes the Task Model introduced earlier. The input to the model is a sequence of bus activity, called a snapshot. The bus activity captured by a snapshot is partitioned into a string of input symbols corresponding to individual bus transfers that are processed sequentially by the FDN. As the input symbols are processed, the FDN models the flow of diagnostic information by the movement of tokens among the information places in the net. When sufficient information is obtained to check a fault, a transition fires to place a token in a diagnostic place to initiate a checking process and record the fact that the fault has been tested. After each snapshot is analyzed, the fault status is defined by the token marking in the model. A detailed description of the model, mathematical properties, and examples are given in reference [10]. A prototype version of the BIT processor [15] was designed to test and verify the models ability to detect most common faults in microprocessor systems. These results demonstrated the FDN models ability to perform on-line fault diagnostics without disturbing the system's operation. The complexity of the FDN model depends on the number and types of faults to be diagnosed. By solving the model for varies input strings (bus activities), the model can be used to throughly test fault diagnostic procedure before system implementation. This is an important application of the model in critical environments.

III. Personnel Supported and Degrees Granted

During the three year duration of the grant, a considerable number of students were supported and many of these students received degrees with research topics stemming from the research sponsored. A detailed list of the students supported follows:

Name	Degree Granted
Glenn Dunn	M.S.E.E.
Hsin Chia Fu	Ph.D.
Tony Stevens	Ph.D.
Mario Chavez	M.S.E.E. (Ph.D. Pending)
Bruce Malm	M.S.E.E. (Ph.D. Pending)
David Comer	B.S.E.E.
Steven Castillo	B.S.E.E.
Bruce Erickson	B.S.E.E.
Jay Lory	B.S.E.E. (Pending)
Martin Small	B.S.E.E. (Pending)
Jay Jordan	Ph.D. (Pending)

Besides these students, the principal investigator, Dr. G.M. Flachs was supported twenty-five percent of the academic year throughout the duration of the grant.

IV. Publications

Several publications, dissertations, and technical reports have resulted from the sponsored research. Several major publications are currently being prepared from the dissertations supported by this grant. These publications will be reported later.

1. 1979 "Real-Time Processing with Multiple Bit-Slice Microprocessors," U.S. Army Workshop on Microprocessors, Pingree Park, Colorado, August 22-24, 1979.

2. 1979 "State-of-the-Art in Real-Time Tracking," U.S.A.F. Workshop on Tracking Systems, U.S.A.F. Academy, Colorado Springs, Sept. 1979.

3. 1980 "High Speed Multiprocessors and Applications," Proceedings of 1980 Joint Automatic Control Conference (JACC-80), Vol. 1, pp. WP4-A1 through WP4-A8, San Francisco, California, Aug. 1980.

4. 1980 "Multi-Intensity Picture Reconstruction from Projections," 1980 Government Electrocircuit Applications Digest (COMAC-80), pp. 93-96, Houston, Texas, Nov. 1980.

5. 1980 "A Communications Unit for Multiprocessing Systems," Glenn A. Dunn, M.S.E.E. Technical Report, New Mexico State University, Las Cruces, New Mexico.

6. 1980 "A Virtual Anything Communications Channel," John Kates, M.S.E.E. Technical Report, New Mexico State University, Las Cruces, New Mexico.

7. 1980 "Interactively Implementing Microprocessor System Designs," Patricia Grubel, M.S.E.E. Technical Report, New Mexico State University, Las Cruces, New Mexico.

8. 1980 "Real-Time Multi-Intensity Image Reconstruction from Projections for Multi-Processor Implementation," Ph.D. Dissertation, Anthony K. Stevens, New Mexico State University, Las Cruces, New Mexico.

9. 1981 "The Intercommunications of a Real-Time Distributive Architecture," Bruce N. Malm, M.S.E.E. Technical Report, New Mexico State University, Las Cruces, New Mexico.

10. 1981 "On-Line Fault Diagnostics of Multiple Microprocessor Systems," Ph.D. Dissertation, Hsin Chia Fu, New Mexico State University, Las Cruces, New Mexico.

11. 1981 "A Statistical Approach to Image Segmentation," Proceedings of Pattern Recognition and Image Proc. Conf., Dallas, Texas, Aug. 1981.

12. 1981 "Real-Time Statistical Transfer for IR Focal Plane Array," Proc. of SPIE's 25th International Symposium, Vol. 302, San Diego, Calif., Aug. 1981.

13. 1981 "Multiprocessor System Design for Real-Time Processor Problems," Proc. of ELECTRO-81 Tercer Seminario de Ingenieria Electronica, Nov. 9-13, 1981.

14 1981 "D Flip/Flop Substracts Frequencies," EDN, p. 199, April 15, 1981.

15. 1981 "Real-Time Fault Diagnostics for Multiple Microprocessor Systems," 1981 Real-Time Systems Symposium, Dec. 1981.

Publications In Preparation

16. "Unique Multi-Intensity Pictures With Respect to Two Projections," Submitted for publications to IEEE Transactions.

17. "Concurrent JCL for Loosely Coupled Distributed System," In preparation.

18. "A Low Cost Multi-Microcomputer Architecture," In preparation.

19. "A Dynamic Model for Multiple Microcomputer Applications," Bruce Malm, Ph.D. dissertation in preparation, New Mexico State University, Las Cruces, New Mexico.

20. "Architectural Design of Multiple Microcomputer Systems," Mario Chavez, Ph.D. dissertation in preparation, New Mexico State University, Las Cruces, New Mexico.

21. "A Dynamic Model for Multiple Microcomputer Communication Systems," Elliot Bergsagel, Ph.D. dissertation currently in progress, New Mexico State University, Las Cruces, New Mexico.

LMED

8